REALSIMPLE Lab on the Doppler Effect, Flanger, and the Leslie

Julius O. Smith III and Nelson Lee,

RealSimple Project[*]
Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305

June 5, 2008

Abstract

This is the REALSIMPLE Doppler, Flanger and Leslie Laboratory. In this lab we will extend the usage of the delay line from the Reverberation Lab to create time-varying effects. Such effects have been prominent in popular recordings from the 1970's to the present day. The Leslie, for example, is a sound associated with organ players around the world. For artist recommendations, search for Joey DeFrancesco, a jazz organ virtuoso and Charlie Hunter, a guitarist who plays through a Leslie rig. In this lab, you will implement digital versions of the Leslie, Doppler and Flanger.

Starter code is available `here`.

1 Doppler

The Doppler Effect is most notably mentioned in the study of physics. It is the effect attributed to the change of pitch of an approaching/passing ambulance with a siren. You will implement the Doppler Effect in the digital realm with a model that is physics-based: a time-varying delay-line.

(50 pts) **Doppler**

1.1 Write Doppler.cpp

Write an STK class called `Doppler.cpp` which simulates the *Doppler effect* using an interpolating time-varying delay line. The inputs to the `tick:` method should be:

- the input signal sample $x(n)$
- a Doppler-shift parameter $\tilde{v}(n) = v_s(n)/c$, which stands for the current speed of the source toward the listener, $v_s(n)$, normalized by sound speed, c.

The formula for the Doppler frequency shift is

$$f(n) = \frac{f_0}{1 - \tilde{v}(n)}$$

[*]Work supported by the Wallenberg Global Learning Network

where f_0 is the frequency emitted by the source at rest.

Write a test program which varies the Doppler parameter sinusoidally as

$$\tilde{v}(n) = 0.1 * \sin(2\pi t)$$

and run the Doppler shifter on a test sinusoid given by

$$x(n) = \sin(2\pi 220 nT), \quad 0 \leq nT \leq 1.$$

where $T = 1/10000$ denotes the sampling period. Listen to the result to make sure it has no artifacts and that the pitch goes up to a maximum, down to a minimum, and back up to the original pitch.

1.2 Using Different Delay Lines

Change `DelayL` to `DelayA` and explain in what ways the performance differs from the previous case.

Turn in your commented program listings and a printout of the first 100 numbers (or so) of your simulation.

2 Leslie

(30 pts) **Leslie**
Look at the STK class `Leslie/Leslie.cpp` given as starter code. It consists of four `Doppler` instances you have just created, each delayed by some amount, and then put through a reverberator. For a general description of the Leslie effect, see
http://www.theatreorgans.com/hammond/faq/mystery/mystery.html and/or
http://www.geofex.com/Article_Folders/lera/lera.htm.

The basic design here is that the direct signal from the main rotating horn gives rise to one of the Doppler units, and the other three correspond to the first three side cabinet reflections. In principle, the reverberator represents the remaining cabinet reflections, plus any desired room reverberation.

Control parameters you can experiment with are the speed and the length of the rotating horn along with the dimension of the cabinet. Optional parameters you may include are the relative phase of the Doppler modulations (e.g., make them uniformly distributed from 0 to 2π as a default) and any reverberation parameters you want to bring out.

2.1 Complete Leslie.cpp

Complete the code in `Leslie.cpp` in the `tick` method.

2.2 Write the STK Test Program

Write a test program that takes any sound file and runs it through the Leslie simulator, varying the speed smoothly from 1 Hz to 10 Hz and back down to 2 Hz over the duration of the sound file.

2.3 Diagram the Model of the Leslie

Draw a diagram of the `Leslie` according to the parameter names and settings in the code, showing the cabinet dimension, horn length and its position at a time instant etc. Also, describe how the multi-path delays are approximated in the simulator. Submit your commented program listing.

3 Flanging

(20 pts) **Flanging**
Look at the STK class
`Flanging.cpp` given as starter code. For more information specific to this assignment, see

`http://ccrma.stanford.edu/~jos/pasp/Flanging.html`.

3.1 Complete Flanging.cpp

Complete the `tick` method in `Flanging.cpp` and set up the parameter values in the constructors to give a nice effect when testing in the main program.

3.2 Write the STK Test Program

Write a test program that takes as input (i) noise generated from an STK class called `Noise` (ii) any sound file, which then goes through the flanging effect.

3.3 Test Your Flanger

Test your flanger on the file `alb.wav` (also available in Ogg Vorbis format as `alb.ogg`.

Submit your commented program listing. Make sure it shows both cases, perhaps with one commented out, of noise input and sound file input.